A Complexity Theory for Parallel Computation

- The questions:
  - What is the computing power of “reasonable” parallel computation models?

    Is it possible to make technology independent statements?
  - Is it possible to relate parallel computations to sequential computations?
  - What are the limits of parallel computations:

    are there algorithmic problems with efficient sequential computations which are hard to parallelize?
- The answers:
  - we relate parallel time and sequential space
  - and define the notion of $\mathcal{P}$-completeness which allows to identify algorithmic problems which are (apparently) hard to parallelize.

Parallel Time

The question: which algorithmic problems with input size n can be solved by a parallel computer in time $T(n)$?

- Observe that we bound the computation time, but not the number of processors.
  - We assume an unlimited supply of processors.
  - Initially only n processors are active.
  - An active processor may activate at most one inactive processor per step.
- Which parallel computer? Any "reasonable" parallel computer, maybe the latest computer generation in 2100!
- Is there a connection between parallel time and sequential space?
  - Intuition: it is possible to simulate a reasonable parallel computer with run time $T(n)$ in sequential space poly$(T(n))$.
  - Can we simulate a sequential computer with space complexity $T(n)$ by a parallel machine with run time poly$(T(n))$?

Space Complexity I

The architecture of an off-line Turing machine:

- an input tape which stores the input $\$w_1 \cdots w_n\$$. The dollar symbol is used as end marker, $w_1, \ldots, w_n$ belong to the input alphabet Σ.
- a work tape which is infinite to the left and to the right. The tape alphabet is binary.
- an output tape which is infinite to the right. Cells store letters from an out alphabet Γ.
- Modifying the tapes:
  - The input tape has a read-only head which can move to the left or right neighbor neighbor of a cell. Initially the head visits the cell storing the leftmost letter of the input.
  - The work tape has a read/write head with identical movement options. Initially all cells store the blank symbol B.
  - The output tape has a write-only head.

Space Complexity II

The computation of an off-line Turing machine M:

- The Turing machine computes by
  - reading its input,
  - modifying its work tape
  - and printing onto its output tape.
- If the machine stops with output 1, then we say that the machine accepts the input; the machine rejects the input, if it writes a 0.
- For Turing machine M and input alphabet Σ, $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$ is the language accepted by M.
- The space complexity of M:
  - $\text{space}_M(w)$ is the number of different cells of the work tape which are visited when computing on w.
  - $\text{space}_M(n) := \max_{w \in \Sigma^n} \text{space}_M(w)$ is the worst-case space complexity of M on inputs of length n.

Space Complexity III

Let $s : \mathbb{N} \to \mathbb{N}$ be a function. Then

$$\mathrm{DSPACE}(s) = \{L \subseteq \{0,1\}^* \mid \begin{array}{l} L = L(M) \text{ for a Turing machine } M \\ \text{with } \mathrm{space}_M(n) = O(s(n)) \end{array} \}$$

is the complexity class of all languages computable in (deterministic) space s.

Important space classes:

- $\mathrm{DSPACE}(0)$ is the class of regular languages.
- $\mathrm{DL} = \mathrm{DSPACE}(\log_2 n)$: deterministic logarithmic space is required to remember an input position.
- $\mathrm{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathrm{DSPACE}(n^k)$ is an extremely powerful complexity class containing $\mathcal{P}$, $\mathcal{NP}$, polynomial time quantum computations ...

How Robust Are Space Complexity Classes?

- Space classes do not change, if we replace Turing machines by reasonable deterministic machine models.
- What about nondeterministic machines?
  - Define $\mathrm{nspace}_M(w)$ as the maximal number of cells of the work tape, which are visited during an accepting computation on input w.
  - $\mathrm{nspace}_M(n) := \max_{w \in \Sigma^n \cap L(M)} \mathrm{nspace}_M(w)$ is the nondeterministic space of M for input size n.
  - Finally: $\mathrm{NSPACE}(s) :=$
    $$\{L \subseteq \{0,1\}^* \mid \begin{array}{l} L = L(M) \text{ for a nondeterministic Turing} \\ \text{machine with } \mathrm{nspace}_M(n) = O(s(n)) \end{array} \}.$$
- Theorem of Savitch: $\mathrm{NSPACE}(s) \subseteq \mathrm{DSPACE}(s^2)$.
- Important space class: $NL = \mathrm{NSPACE}(\log_2 n)$, all languages recognizable in nondeterministic logarithmic space.

The Computation Graph of A Nondeterministic TM

Let M be a nondeterministic off-line Turing machine with space complexity at most s. Let w be an input of M.

- A configuration consists of the contents of the work tape, the current state and positions of the input and work tape heads.
- The computation graph $G_M(w)$ of M on input w:
  - The configurations on input w are the nodes of $G_M(w)$.
  - An edge (k_1, k_2) from configuration k_1 to configuration k_2 belongs to $G_M(w)$ iff configuration k_2 is a successor of k_1.
  - An additional node yes is added and, for any accepting configuration k, edges (k, yes) are inserted.
- M accepts input w iff there is a path in $G_M(w)$ from the initial configuration k_0 to node yes.
- If M uses space $s(n)$, then M has at most $2^{O(s(n))} \cdot O(1) \cdot (n+2) \cdot s(n) = n \cdot 2^{O(s(n))}$ configurations.

$\mathcal{NC}$ and Logarithmic Space

- DL $\subseteq$ NL
- and any language in NL can be accepted in time $O(\log_2^2 n)$ with a polynomial number of processors.

- DL $\subseteq$ NL: obvious.
- Let L be an arbitrary language in NL.
  - Let $L = L(M)$ for a nondeterministic Turing machine using logarithmic space.
  - $G_M(w)$ has $N = \text{poly}(|w|)$ nodes and can be constructed in logarithmic time with a polynomial number of processors:

    reserve one processor for any pair of configurations to check, whether the pair corresponds to a transition.
  - M accepts w iff there is a path in $G_M(w)$ from the initial configuration k_0 to the unique accepting configuration yes.
- We need a real fast solution of the transitive closure problem: $O(\log_2^2 N)$ time with $\text{poly}(N)$ processors will do.

The Transitive Closure Problem Revisited

Determine the transitive closure of a graph G with N nodes.
Use circuits.

- We have parallelized Warshall's algorithm, but its speed is insufficient.
- Instead we use boolean matrix multiplication.
  - Let A be the adjacency matrix of G. Set all diagonal entries to one (i.e., insert self loops) and call the new matrix B.
  - $B[i,j] = 1$ iff there is a path of length one from i to j.
  - And inductively, $B^{r+1}[i,j] = \bigvee_{k=1}^{N} B^r[i,k] \wedge B[k,j]$: $B^{r+1}[i,j] = 1$ iff there is a path of length r from i to k and an edge from k to j.
  - There is a path from i to j iff $B^{N-1}[i,j] = 1$.
- Compute $B, B^2, \ldots, B^{2^j}, B^{2^{j+1}}, \ldots, B^{2^t}$ for $t = \lceil \log_2 N \rceil$.
- $t = \lceil \log_2 N \rceil$ matrix products. Per matrix product depth $O(\log_2 N)$ and size $O(N^3)$. All in all: depth $O(\log_2^2 N)$ and size $O(N^3 \cdot \log_2 N)$.

Uniform Circuit Families

We consider $\{\neg, \wedge, \vee\}$-circuits as a parallel computing model.

- Depth corresponds to parallel time.
- Is there a connection to sequential space?

- To apply circuits to inputs of various input length, we have to work with families $(S_n \mid n \in \mathbb{N})$ of circuits S_n for input size n.
- We should require that the description of the family is only moderately complex.
  - We say that the circuit family is uniform if, given input 1^n, its circuit S_n can be described by a deterministic off-line Turing machine M with space complexity $O(\log_2 \text{size}(S_n))$.
  - M has to eventually print all edges of S_n onto its output tape as well as all nodes with their assigned operation, resp. with their assigned input bit.

Parallel Time and Sequential Space I

Assume that $s(n) = \Omega(\log_2 n)$. A nondeterministic off-line Turing machine M with space complexity $s(n)$ can be simulated by a uniform circuit family $(S_n \mid n \in \mathbb{N})$ in time $O(s^2(n))$.

- Input w belongs to $L(M)$ iff there is a path in $G_M(w)$ from the initial configuration k_0 to the accepting configuration yes.
- Let N be the number of nodes of $G_M(w)$.
  - Compute the transitive closure of $G_M(w)$ in depth $O(\log_2^2 N)$.
  - We are required to build a uniform circuit family.
    - Configurations have length $O(\log_2 n + s(n))$
    - and space $O(\log_2(n) + s(n))$ is sufficient to output all valid transitions between configurations on input 1^n.
- We are done, since $N = 2^{O(s(n))}$ and $\log_2^2 N = s^2(n)$.

Parallel Time and Sequential Space II

A uniform circuit family $(S_n \mid n \in \mathbb{N})$ of depth $s(n)$ can be simulated by a deterministic off-line Turing machine M within space $O(s^2(n))$.

- How does M work?
- Set $n = |w|$. Since S_n has depth $s(n)$, the circuit has at most $2^{s(n)+1} - 1$ nodes.
  - Evaluate S_n via depth-first search with a stack of height $O(s(n))$.
  - Since the stack holds nodes from the circuit, all we need is space complexity $O(s^2(n))$.

- Indeed, already space $O(s(n))$ suffices. Why?
- We have shown the Theorem of Savitch:
  - A nondeterministic Turing machine with space $s(n)$ can be simulated by a uniform circuit family of depth $O(s^2(n))$.
  - The uniform circuit family can be simulated by a deterministic Turing machine with space $O(s^2(n))$.

The Parallel Computation Thesis

- We have found a quadratic relationship between the depth of circuits and the space complexity of Turing machines.
- The depth of circuit families and the computing time of PRAMs is also polynomially related.
- The following conjecture (parallel computation thesis) is therefore natural:

  > parallel time for any *reasonable* parallel computing model is polynomially related to the space complexity for sequential computations.

- The thesis is mainly of theoretical interest, since the size of the parallel computing model is not restricted.
- However, when trying to design a parallel algorithm,
  - first check if low-space algorithms exist.
  - If not, then there may be only little speedup possible.

The Class $\mathcal{NC}$

$\mathcal{NC}$ is the class of all problems with parallel algorithms

- running in time poly($\log_2 n$)
- with a polynomial number of processors.

- $\mathcal{NC}$ is a subset of $\mathcal{P}$: each parallel step can be simulated in polynomial time.
- As a consequence $\mathcal{NC}$ is the class of problems with
  efficient sequential algorithms and drastic parallel speedups.

Which Algorithmic Problems Belong To $\mathcal{NC}$?

- Many important problems of linear algebra:
  - the matrix-vector product,
  - the matrix-matrix product,
  - computing the determinant and solving linear systems,
  - and the Fast Fourier Transform.
- Many dynamic programming problems such as:
  - the transitive closure problem,
  - the all-pairs-shortest path problem,
  - pairwise alignment
  - and the RNA secondary structure prediction.
- computing connected components and minimum spanning trees.
- Sorting.
- All context-free languages (and hence all regular languages).

Which Problems Probably Do Not Belong To $\mathcal{NC}$?

- Which languages in $\mathcal{P}$ do not admit "extreme" speedups?
- Important observation: there are hardest languages $L \in \mathcal{P}$:
  - if L has real fast parallel algorithms with polynomially many processors, then all problems in $\mathcal{P}$ have fast algorithms with polynomially many processors.
  - Hence, in all likelihood a hardest language has no extreme speedups.
- The roadmap:
  - Define reductions to compare languages with respect to their potential speedup.
  - Show that many important hardest languages exist.

Reductions

L_1 is reducible to L_2 ($L_1 \leq_{\text{par}} L_2$), iff there is a "translation" T with

$$w \in L_1 \Leftrightarrow T(w) \in L_2.$$

T has to be computable in poly-logarithmic time with a polynomial number of processors.

- Assume $L_1 \leq_{\text{par}} L_2$ and that L_2 belongs to $\mathcal{NC}$.
- The reduction should compare languages with respect to achievable speedups. Does L_1 also belong to $\mathcal{NC}$?
  - run the translation T on input w to compute $T(w)$ in poly-logarithmic time with a polynomial number of processors.
  - Since $L_2 \in \mathcal{NC}$, determine whether $T(w)$ belongs to L_2 in poly-logarithmic time with a polynomial number of processors.
  - Accept w for language L_1 iff $T(w) \in L_2$.
- And L_1 also belongs to $\mathcal{NC}$.

$\mathcal{P}$-complete Problems: The Concept

- A language K is $\mathcal{P}$-hard iff $L \leq_{\text{par}} K$ for all languages $L \in \mathcal{P}$.
- K is $\mathcal{P}$-complete iff K is $\mathcal{P}$-hard and K belongs to $\mathcal{P}$.

What happens, if a $\mathcal{P}$-hard language K belongs to $\mathcal{NC}$?

- We show that all languages in $\mathcal{P}$ have "extreme" speedups.
- Let $L \in \mathcal{P}$ be arbitrary.
  - $L \leq_{\text{par}} K$, since K is $\mathcal{P}$-hard.
  - But then $L \in \mathcal{NC}$, as we have just seen, since K belongs to $\mathcal{NC}$.

If some $\mathcal{P}$-hard language belongs to $\mathcal{NC}$, then $\mathcal{P} = \mathcal{NC}$ and all problems in $\mathcal{P}$ have extreme speedups.

How To Show $\mathcal{P}$-Completeness?

- If $L_1 \leq_{\text{par}} L_2$ and $L_2 \leq_{\text{par}} L_3$, then $L_1 \leq_{\text{par}} L_3$.
- If K is $\mathcal{P}$-hard and $K \leq_{\text{par}} L$, then L is $\mathcal{P}$-hard.

- Why is the reduction transitive?
  - If $L_1 \leq_{\text{par}} L_2$ holds because of translation T_1
  - and if $L_2 \leq_{\text{par}} L_3$ holds because of translation T_2,
  - then $L_1 \leq_{\text{par}} L_3$ holds because of translation $T_2 \circ T_1$.
- Assume that K is $\mathcal{P}$-hard and that $K \leq_{\text{par}} L$ holds.
  - Let $M \in \mathcal{P}$ be arbitrary. Then $M \leq_{\text{par}} K$, since K is $\mathcal{P}$-hard.
  - Moreover $K \leq_{\text{par}} L$ holds. And $M \leq_{\text{par}} L$ follows by transitivity.
  - Hence L is $\mathcal{P}$-hard.

The Circuit Value Problem

Let S be a circuit with a single output gate.

Determine whether S accepts an input w.

I.e., CVP $= \{(S, w) \mid$ the circuit S accepts $w\ \}$.

- CVP is an easy problem for sequential computation:

  evaluate a circuit with, say, depth-first search.

  Hence CVP $\in \mathcal{P}$.
- However a parallel evaluation in poly-logarithmic time seems hard for circuits of large depth.

Show that CVP is $\mathcal{P}$-complete.

Proof Outline

Show $L \leq_{\text{par}} \text{CVP}$ for an arbitrary language $L \in \mathcal{P}$.

- What do we know about L?

  There is a Turing machine M with a single tape which computes in time $t(n) = O(n^k)$ and

  $$w \in L \Leftrightarrow M \text{ accepts } w.$$

- The goal: simulate M by a circuit family $(S_n \mid n \in \mathbb{N})$ of polynomial size such that

  $$M \text{ accepts } w \Leftrightarrow S_{|w|} \text{ accepts } w.$$

- Set $T(w) = (S_{|w|}, w)$. Then

  $$w \in L \Leftrightarrow M \text{ accepts } w \Leftrightarrow S_{|w|} \text{ accepts } w \Leftrightarrow (S_{|w|}, w) \in \text{CVP}.$$

How to choose $S_{|w|}$?

The Two-Dimensional Structure of $S_{|w|}$

Assume that w is input for M and that w has length n.

- M runs in time $t(n)$ and hence it visits only the cells with addresses $-t(n), -t(n)+1, \cdots, -1, 0, +1, \cdots, t(n)$.
  - Here we assume that letter w_i is written on the cell with address i
  - and that the read/write head of M initially "sits" on the cell with address 0.
- We simulate M on input w with a circuit S_n which is built like a two-dimensional mesh.
  - The ith "row" of S_n reflects the configuration of M on input w at time i and its nodes correspond to the tape cells,
  - remember the current state of M and indicate the current position of the read/write head.
- The circuit has to compute the configuration at time $i+1$ from the configuration at time i.

Updating Configurations

- We work with small subcircuits $S_{i,j}$ which are responsible for cell j of the tape at time i.
  - Subcircuit $S_{i+1,j}$ "asks" subcircuit $S_{i,j}$ if the head is visiting cell j.
  - If the answer is negative, then $S_{i+1,j}$ assigns the letter remembered by $S_{i,j}$ to cell j.
  - If the answer is positive:
    - $S_{i,j}$ also stores the current state which it transmits together with the current contents of cell j to $S_{i+1,j}$.
    - $S_{i+1,j}$ determines the new contents of cell j, the new state and the neighbor which will be scanned at time $i+1$.
- All subcircuits $S_{i,j}$ are identical except for the subcircuits $S_{0,j}$, since cell j stores w_j for $1 \leq j \leq n$ and the blank symbol otherwise.
- A binary tree on top of row $t(n)$ has to be added to check whether the final state is accepting.
- S_n can be constructed in logarithmic time with polynomially many processors.

Difficult Variants of CVP

- In M-CVP we assume that S is a monotone circuit and ask whether S accepts input w.

  A circuit is monotone, if it does not have negations.

- M_2-CVP is defined as M-CVP, except that the circuit is required to have fan-out at most two.
- In NOR-CVP the circuit is built from NOR-gates only.

  Remember that $\mathrm{NOR}(u, v) = \neg(u \vee v)$.

We show that all variants are $\mathcal{P}$-complete.

- All three problems belong to $\mathcal{P}$.
- CVP $\leq_{\mathrm{par}}$ M-CVP, M-CVP $\leq_{\mathrm{par}}$ M_2-CVP and CVP $\leq_{\mathrm{par}}$ NOR-CVP.

M-CVP: Pushing Negations to the Sources I

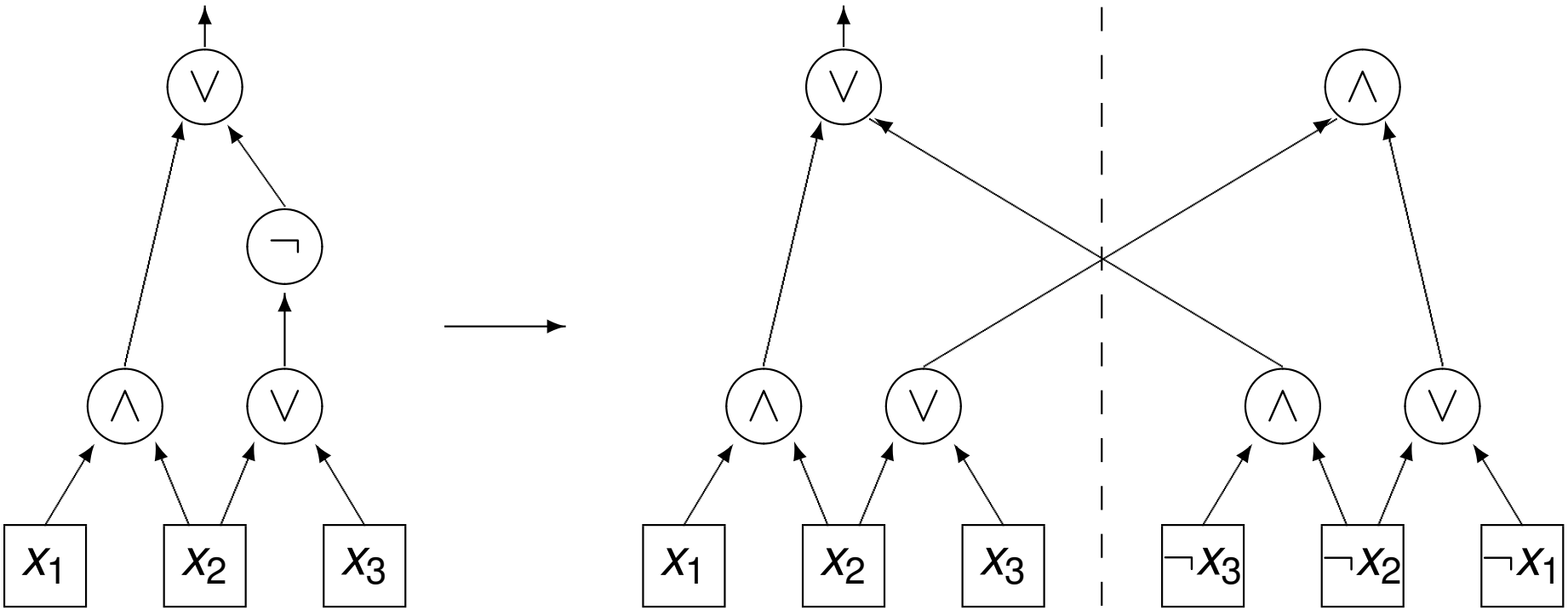
S is a $\{\neg, \wedge, \vee\}$-circuit.

Is there an equivalent circuit S' which uses negations only for input bits?

- Push negations to the inputs using the De'Morgan rules
  $$\neg(x \wedge y) \equiv \neg x \vee \neg y \quad \text{and} \quad \neg(x \vee y) \equiv \neg x \wedge \neg y.$$
- However there is a problem:
  - The result of a gate may be needed in its negated as well as in its unnegated form.
  - The equivalent circuit S' should provide both results.
- S' will at most double in size. Why?

M-CVP: Pushing Negations to the Sources II

- The circuit S' has two gates u_{pos} and u_{neg} for any gate u of S.
  - If u is an input bit x_i, then $u_{\text{pos}} = x_i$ and $u_{\text{neg}} = \neg x_i$.
  - u_{pos} keeps the functionality of u, u_{neg} interchanges $\wedge$ and $\vee$.
- For any edge (u, v) of S, the equivalent circuit S' has two edges, namely $(u_{\text{pos}}, v_{\text{pos}})$ and $(u_{\text{neg}}, v_{\text{neg}})$.



M-CVP is $\mathcal{P}$-Complete

For any circuit S there is an equivalent circuit S' with negations only for inputs. S' can be constructed in logarithmic time from S. The size of S' at most doubles in comparison to S.

- Let (S, w) be an input for CVP.
  - Construct the circuit S', remove all negations from S' and call the resulting monotone circuit S_m.
  - We have to modify the input for S_m, since we removed negations:
    - The input for a not-negated input gate is left unchanged,
    - whereas the input for a negated input gate is flipped.
    - Call the new input w_m.
  - Define the translation $T(S, w) = (S_m, w_m)$. Then

$$\begin{aligned}(S, w) \in \text{CVP} &\Leftrightarrow S \text{ accepts } w \Leftrightarrow S_m \text{ accepts } w_m \\ &\Leftrightarrow (S_m, w_m) \in \text{M-CVP}.\end{aligned}$$

- We have verified the reduction CVP $\leq_{\text{par}}$ M-CVP, since T can be computed in logarithmic time.

M_2-CVP and NOR-CVP are $\mathcal{P}$-Complete

- The construction of the reduction M-CVP $\leq_{\text{par}}$ M_2-CVP is left as an exercise.
- To show: CVP $\leq_{\text{par}}$ NOR-CVP.
  - NOR is a basis, since
    - $\neg u \equiv \text{NOR}(u, u)$
    - $u \wedge v \equiv \text{NOR}(\neg u, \neg v)$
    - and $u \vee v \equiv \text{NOR}(\text{NOR}(u, v), \text{NOR}(u, v))$.
  - If (S, w) is an input of CVP:
    - replace all gates by small NOR-circuits to obtain an equivalent NOR-circuit S^*.
    - The translation $(S, w) \mapsto (S^*, w)$ establishes the wanted reduction to NOR-CVP.

Linear Programming

Minimize a linear objective function $c^T \cdot x = \sum_{i=1}^{n} c_i x_i$ over the real numbers subject to

- linear constraints $\sum_{j=1}^{n} A[i,j] \cdot x_j \geq b_i$ for $j = 1, \ldots, m$
- and to non-negativity constraints $x_1 \geq 0, \ldots, x_n \geq 0$.

In shorthand:

$$\min c^T \cdot x \text{ s.t. } A \cdot x \geq b \text{ and } x \geq 0,$$

- A is the $m \times n$ constraint matrix with $A = (A[i,j])_{1 \leq i \leq m, 1 \leq j \leq n}$,
- $c = (c_1, \ldots, c_n)$ is the vector of coefficients of the objective function
- and $b = (b_1, \ldots, b_m)$ is the vector of right hand sides.

Linear Programming: What Is Known?

- Efficient sequential solutions exist, namely the ellipsoid method or interior point methods.
- May be the most powerful optimization method with efficient algorithms. For instance the bipartite matching problem for the graph G:
  - $\max \sum_{e \in E} y_e$ s.t. $A_G \cdot y \leq 1$ and $y \geq 0$.
  - The incidence matrix A_G of G has a row for every node $u \in V_0 \cup V_1$ and a column for every edge $e \in E$: $A_G[u, e] = 1$ iff node u is an endpoint of edge e and $A_G[u, e] = 0$ otherwise.
  - Although fractional solutions are allowed, all vertices of the polytope are integral.
- We show that the general linear programming problem is $\mathcal{P}$-complete. However fast parallel algorithms exist, if all parameters (A, b and c) are nonnegative.

Linear Programming As A Decision Problem

- The Linear Inequalities problem (LIP):
  - We are given an $m \times n$ matrix $A \in \mathbb{Z}^{m \cdot n}$ and a vector $b \in \mathbb{Z}^m$.
  - The pair (A, b) belongs to LIP iff the system $A \cdot x \geq b$ of linear inequalities has a solution.
- The Linear Programming Problem (LPP)
  - We receive the same input as in LIP, but obtain additionally a vector $c \in \mathbb{Z}^n$ and a rational number α.
  - The quadruple (A, b, c, α) belongs to LPP iff there is a vector x that solves the linear system $A \cdot x \geq b$ and also satisfies $c^T \cdot x \leq \alpha$.
  - LPP is the language version of the Linear Programming Problem:

    minimize the linear function $c^T \cdot x$ subject to the linear constraints $A \cdot x \geq b$.

Linear Inequalities are $\mathcal{P}$-Complete

Show the reduction M-CVP $\leq_{\text{par}}$ LIP.

- Let (S, w) be an input of M-CVP.
  Assign linear inequalities to any node c of the monotone circuit S.
  - c is a source storing the ith input bit: use the inequalities
    $x_c \geq 0,\ -x_c \geq 0$, if $w_i = 0$ and $x_c \geq 1,\ -x_c \geq -1$, if $w_i = 1$.
  - $c \equiv a \wedge b$ is an AND-gate: use the inequalities
    $x_a - x_c \geq 0, x_b - x_c \geq 0, x_c - x_a - x_b \geq -1, x_c \geq 0$.
  - $c \equiv a \vee b$ is an OR-gate: now choose
    $x_c - x_a \geq 0, x_c - x_b \geq 0, x_a + x_b - x_c \geq 0,\ -x_c \geq -1$.
- Show by induction on the depth of a gate c of S:
  $x_c =$ the value of gate c for input w.
- Finally, if t is the output gate of the circuit: add the inequality $x_t \geq 1$ to the linear system and

$$(S, w) \in \text{CVP} \Leftrightarrow \text{ S accepts } w \Leftrightarrow \text{ the linear system is solvable.}$$

Linear Programming is $\mathcal{P}$-Complete

Show the reduction LIP $\leq_{\text{par}}$ LPP.

- Choose the translation $(A, b) \mapsto (A, b, \vec{0}, 0)$.

  The existence of a solution x with $A \cdot x \geq b$ and $0^T \cdot x \geq 0$ is equivalent to the existence of a solution of $A \cdot x \geq b$.

- Good parallel algorithms exist for positive linear programs:
  - minimize $c^T \cdot x$ such that $A \cdot x \geq b$ and $x \geq 0$.
  - The constraint matrix A, as well as the right hand side b and the coefficient vector c of the objective function are non-negative.

Network Flow

- An input for the flow problem consists of
  - a directed graph $G = (V, E)$,
  - two distinguished nodes, the source $s \in V$ and the sink $t \in V$
  - and a capacity function $c : E \to \mathbb{R}$.
- A function $f : E \to \mathbb{R}$ is a flow, if $0 \leq f(e) \leq c(e)$ for all edges $e \in E$ and if flow conservation

$$\sum_{u \in V, (u,v) \in E} f(u, v) = \sum_{u \in V, (v,u) \in E} f(v, u)$$

holds for any node $v \in V \setminus \{s, t\}$:

the amount of flow entering v equals the amount of flow leaving v. Flow conservation is not required for the source or the sink.

- A flow f is maximal, if f maximizes

$$|f| = \sum_{u \in V, (s,u) \in E} f(s, u) - \sum_{u \in V, (u,s) \in E} f(u, s),$$

i.e., if the net flow pumped out of s is maximal.

The Flow Problem

- The integrality property holds: if all capacities are integral, then there is a maximal flow which is integral.
- As a consequence, the bipartite matching problem

  determine a largest collection of node-disjoint edges for a given bipartite graph $B = (V_0 \cup V_1, E)$

  is a special case of the flow problem:
  - Add a source s and a sink t to B.
  - Connect s to all nodes in V_0 and all nodes in V_1 to t by edges of capacity one.
  - All edges of B receive capacity one.

The Flow Problem FP

An input (G, s, t, c) belongs to FP iff all capacities are integral and the maximal flow is odd.

M_2-CVP $\leq_{\text{par}}$ FP

Let (S, w) be an input for M_2-CVP and let $G = (V, E)$ be the directed graph of the monotone circuit S.

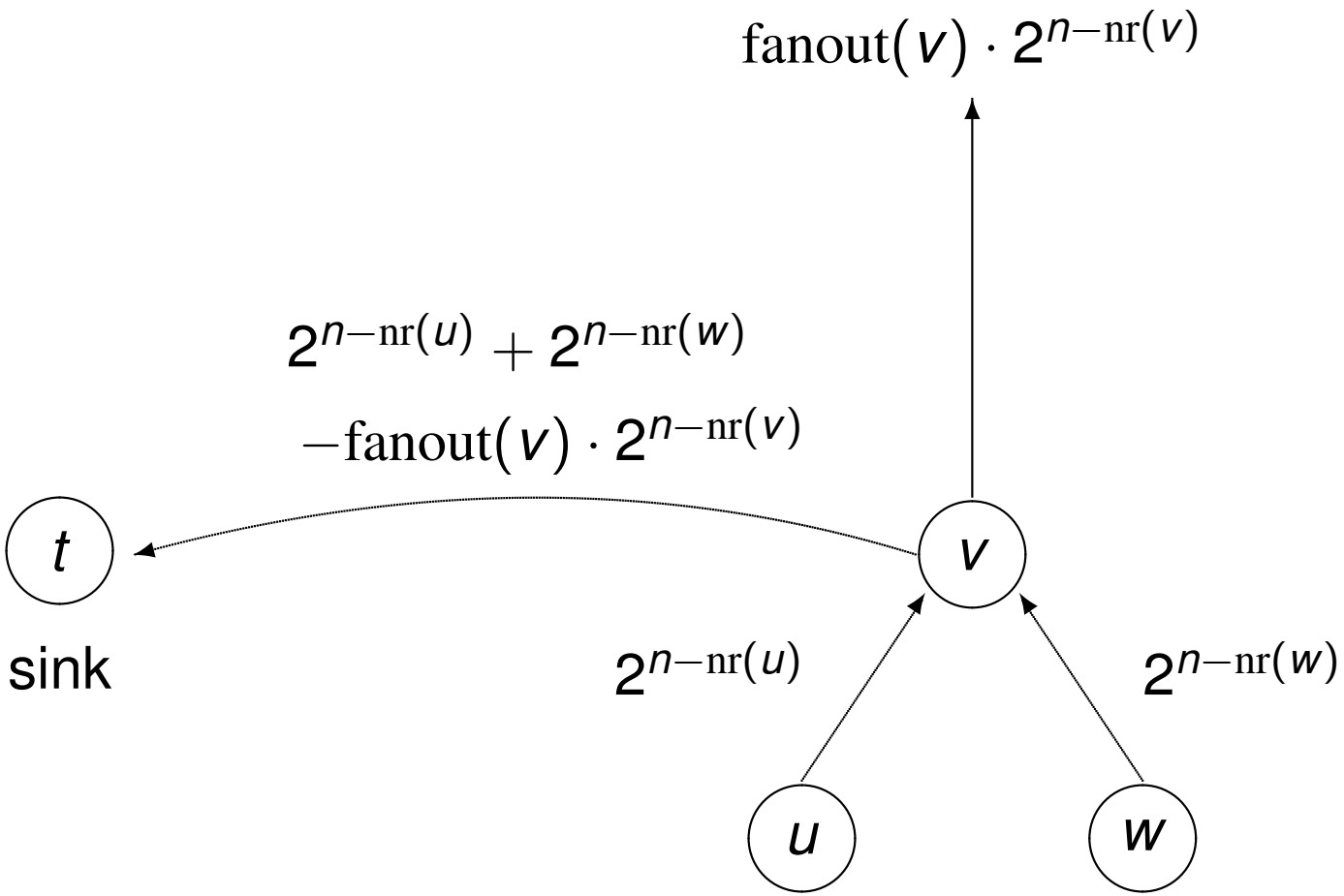
- Assumptions on S:
  - the output gate t_G of S is an OR-gate: if not, then replace S by two copies of S which are fed into an OR-gate.
  - each input node of G has fan-out one: otherwise insert additional copies.
  - All other nodes have fanout at most two.
- The graph G^* for FP:
  - add two new nodes s and t to G.
  - The source s is connected to all input gates of G and the output gate t_G of S is connected to the sink t.
  - Finally add edges from OR-gates to s as well as edges from AND-gates to t.
- fanout(z) counts the number of edges of G leaving z.
  However set fanout(t_G) $= 1$ to account for the edge (t_g, t).

Defining Capacities

- Assume that G has exactly n nodes.
- Determine a topological sort $\mathrm{nr} : V \to \{1, \ldots, n\}$ of G.

- If z is an input node for variable x_i, then
  $c(s,z) = \begin{cases} 2^{n-\mathrm{nr}(z)} & w_i = 1, \\ 0 & \text{otherwise.} \end{cases}$
- If z is an interior node with immediate predecessors x and y: then $c(x,z) = 2^{n-\mathrm{nr}(x)}$ and $c(y,z) = 2^{n-\mathrm{nr}(y)}$.
  - If z is an AND-gate: $c(z,t) = 2^{n-\mathrm{nr}(x)} + 2^{n-\mathrm{nr}(y)} - \mathrm{fanout}(z) \cdot 2^{n-\mathrm{nr}(z)}$,
  - If z is an OR-gate: $c(z,s) = 2^{n-\mathrm{nr}(x)} + 2^{n-\mathrm{nr}(y)} - \mathrm{fanout}(z) \cdot 2^{n-\mathrm{nr}(z)}$.
- Set $c(t_g, t) = 1$.
  - Observe $c(t_G, s) = 2^{n-\mathrm{nr}(x)} + 2^{n-\mathrm{nr}(y)} - 1$.
  - If t_G receives the maximal possible flow $2^{n-\mathrm{nr}(x)} + 2^{n-\mathrm{nr}(y)}$, then the edge (t_G, t) can transport the remaining flow of one to t.
  - All other nodes have links to t of even capacity.

Capacities Of An AND-Gate



The Maximal Flow f^*

Claim: The translation $(S, w) \mapsto (G^*, s, t, c)$ establishes a reduction from M_2-CVP to FP.

Define a flow f^*:

- If z is an input gate: set $f^*(s, z) = c(s, z)$.
  - A flow of $2^{n-\mathrm{nr}(z)}$ is pumped into z whenever the input bit of gate z is one and a zero flow otherwise.
  - There is one edge leaving z which has to be filled to its capacity $2^{n-\mathrm{nr}(z)}$.
- If z is an interior gate which evaluates to one: fill its fan-out(z) many leaving G-edges to capacity and push the excess flow to s respectively to t.

  If t_G "fires", then push flow one along edge (t_G, t).
- If z is an interior gate which evaluates to zero: push zero flow across its leaving G-edges and any excess flow to s respectively to t.

  If t_G does not fire, then push flow zero along edge (t_G, t).

The Cut

Partition the nodes of G^* into the sets

$$V_1 = \{s\} \cup \{z \in G | \text{the gate } z \text{ evaluates to one } \}$$
$$V_0 = \{t\} \cup \{z \in G | \text{the gate } z \text{ evaluates to zero } \}.$$

- all $V_1 \to V_0$ edges of G^* are filled to capacity:
  - G-edges from a node in V_1 to a node in V_0 are filled to capacity by definition of f^*.
  - Edges from AND-gates in V_1 to t are also filled to capacity,
  - whereas edges from OR-gates in V_1 to s stay in V_1.
  - The edge (t_G, t) is also filled to capacity, provided t_G fires.
- All $V_0 \to V_1$ edges of G^* carry flow zero:
  - G-edges from a node in V_0 to a node in V_1 carry flow zero.
  - Edges from AND-gates in V_0 to t stay within V_0.
  - An OR-gate in V_0 receives zero flow along its two G-edges and hence an OR-gate in V_0 has no flow to distribute.
  - The edge (t_G, t) stays inside V_0, if $t_G \in V_0$,

Proof of Optimality

$$|f^*| = \sum_{x \in V_1, y \in V_0, (x,y) \in E} c(x, y).$$

- $|g| \leq \sum_{x \in V_1, y \in V_0, (x,y) \in E} c(x, y)$ for any flow g, since $s \in V_1$ and $t \in V_0$.
- f^* is a maximal flow. As a consequence:
  - if t_G fires, then $|f^*|$ is odd, since all edges carry even flow except for the edge (t_G, t) which carries flow 1.
  - If t_G does not fire, then $|f^*|$ is even, since all edges carry even flow.

FP is $\mathcal{P}$-Complete

$$
\begin{aligned}
(S, w) \in \mathrm{M}_2\text{-CVP} \quad &\Leftrightarrow \quad S \text{ is a monotone circuit with fanout two} \\
&\quad\ \text{and } S \text{ accepts } w \\
&\Leftrightarrow \quad \text{the maximal flow in } (G^*, s, t, c) \text{ is odd} \\
&\Leftrightarrow \quad (G^*, s, t, c) \in FP.
\end{aligned}
$$

- The translation $(S, w) \to (G^*, s, t, c)$ establishes the reduction $\mathrm{M}_2\text{-CVP} \leq_{\mathrm{par}} \mathrm{FP}$.
- Is it necessary to use capacities of exponentional size? Yes.

  FP with capacities of polynomial size can be solved with a randomized $\mathcal{NC}$-algorithm.

Lexicographically First Maximal Independent Set

Determine a maximal independent set for an undirected graph $G = (\{1, \ldots, n\}, E)$:

$I := \emptyset$.

for $v = 1$ to n do
    If (v is not connected with a node from I) then
        $I = I \cup \{v\}$.

- I is a maximal independent set: any node v belongs either to I or is connected with an edge to a node in I.
- In the lexicographically first maximal independent set problem (LFMIS) we are to determine if a node v belongs to the independent set I computed by the for-loop.
  - LFMIS is trivial sequentially, since we only have to run the for-loop.
  - We show that LFMIS is $\mathcal{P}$-complete and hence parallelizing for-loops is non-trivial.

The All-Pairs-Longest-Path Problem

Solve the all-pairs-longest-path problem for a directed acyclic graph with n nodes in time $O(\log_2^2 n)$ with n^3 processors.

- Reduce the all-pairs-longest-path problem to matrix multiplication:
  - Define a non-standard matrix multiplication for $n \times n$ matrices

$$X * Y[u, v] = \max_{w \in V} X[u, w] + Y[w, v].$$

    max replaces addition and addition replaces multiplication.
  - Let A be the adjacency matrix of G and assume that the diagonal of A is zero, i.e., G has no self-loops. Then

$$A^k[u, v] = \max_{w_1, \ldots, w_{k-1}} A[u, w_1] + A[w_1, w_2] + \cdots + A[w_{k-1}, v].$$

- Since G is acyclic, no node w_i is traversed twice and $A^{n-1}[u, v]$ is the length of a longest path from u to v.

Topological Sort

- For the all-pairs-longest path problem:
  - To compute A^{n-1}: compute the matrix powers $A^2, \ldots, A^{2^i}, A^{2^{i+1}}, \ldots, A^{2^t}$ for $t = \lceil \log_2 n \rceil$.
  - The all-pairs-longest path problem can be solved in time $O(\log_2^2 n)$ with n^3 processors.
- The topological sort of a directed acyclic graph G is a bijection $\pi : V \to \{1, \ldots n\}$ with $\pi(u) < \pi(v)$ for any edge (u, v).
  - Let length(v) be the length of a longest path ending in v.
  - Replace each node v by the pair (length(v), v) and sort all pairs.
  - All in all, time $O(\log_2^2 n)$ with n^3 processors is sufficient to perform a topological sort.

LFMIS is $\mathcal{P}$-Complete

Construct the reduction NOR-CVP $\leq_{\text{par}}$ LFMIS.

- Let (S, w) be an input for NOR-CVP and let G be the directed graph for the NOR-circuit S.
- Important observation: all gates with value 1 form an independent set. Why?

  A NOR-gate NOR(u, v) "fires" iff $u = v = 0$.
- The idea:
  - modify the graph G i.e., rename nodes, to obtain a graph G^* such that the for-loop picks all nodes with value 1.
  - Then ask whether the output gate t of S belongs to I.
  - Done, since
    $(S, w) \in$ NOR-CVP $\Leftrightarrow$ t fires $\Leftrightarrow t \in I \Leftrightarrow (G^*, t) \in LFMIS$,
    provided the translation $(S, w) \to (G^*, t)$ can be computed in poly-logarithmic time.

The Translation $(S, w) \to (G^*, t)$

Determine a topological sort for the graph G of S.

- To determine the translation:
  - replace the name of a node by its rank within the topological sort.
  - Thus the smallest number is given to a source which may however evaluate to zero. But the node with smallest number will belong to the independent set!
  - Invent a new node 0, connect 0 with all sources which evaluate to 0 and disregard edge direction. Let G^* be the new (undirected) graph.
- What does the for-loop do initially:
  - The new node 0 is included into I and automatically all sources with value 0 will be disregarded.

Does the reduction work?

Proof of Correctness

The for-loop determines the independent set
$I = 0 \cup \{z \mid z \text{ evaluates to one }\}$.

- We show the claim by induction on the depth of a node. The claim is correct, if $z = 0$ or if z is a previous source of G.
- If $z = \mathrm{NOR}(u, v)$ and z fires:
  - $u = v = 0$, since $z = 1$.
  - By induction hypothesis $u \notin I$ and $v \notin I$.
  - z is included into I.
- If $z = \mathrm{NOR}(u, v)$ and z does not fire:
  - $u = 1$ or $v = 1$ since $z = 0$.
  - By induction hypothesis $u \in I$ or $v \in I$
  - and hence z will be left out.